EXHIBIT G

Retired Document

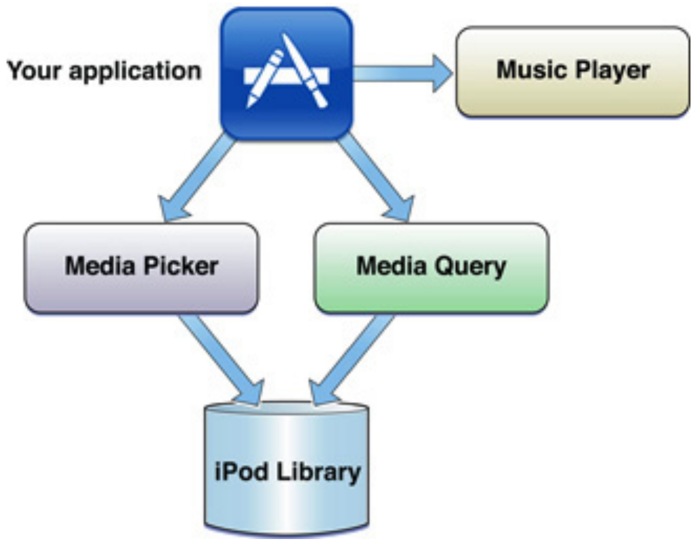
Important: This document may not represent the best practices for current deployment. Links to downloads and other resources may no longer be valid. SeeMediaPlayer Framework API for the latest information.

ce iPod library. The iPod library is the set of media items on a device that a user has synced from iTunes on the desktop.

As shown in Figure 1-1, your application has two ways to retrieve items. The media picker, shown on the left, is an easy-to-use, pre-packaged view controller that behaves like the built-in iPod application's music selection interface. For many applications, this is sufficient.

If the picker doesn't provide the specialized access control that you want, the media query interface—shown to the lower right of your application in the figure—will. It supports predicate-based specification of items from the device iPod library.

Figure 1-1 Using iPod library access



As depicted to the right of your application in the figure, you then play the retrieved media items using the music player provided by this API.

As you can see, the classes in this API address two, usually distinct areas of development: database access—for finding media items to play—and music playback. However, you do not need to be a subject expert in either area to use this API effectively. iPod library access does the "heavy lifting" for you.

For example, the entirety of your music playback code is a single line—whether playing DRM-encrypted AAC files, Apple Lossless tracks ripped from a CD, audio podcasts encoded with iLBC, or audio books. The system automatically sets up buffering, chooses the appropriate codec, and sends the audio directly to the device output hardware.

Using a music player and the media item picker, you can implement music selection and playback without writing any code to access the iPod library. The database access classes provide a complete query system when you need it, and stay out of your way when you don't.

Note: iPod library access applies only to audio-based media items. You cannot play video podcasts, movies, or television shows from the iPod library.

About Music Players

Your application uses a music player in a way that's similar to an end user operating the built-in iPod application. You can programmatically play, pause, seek, and so on.

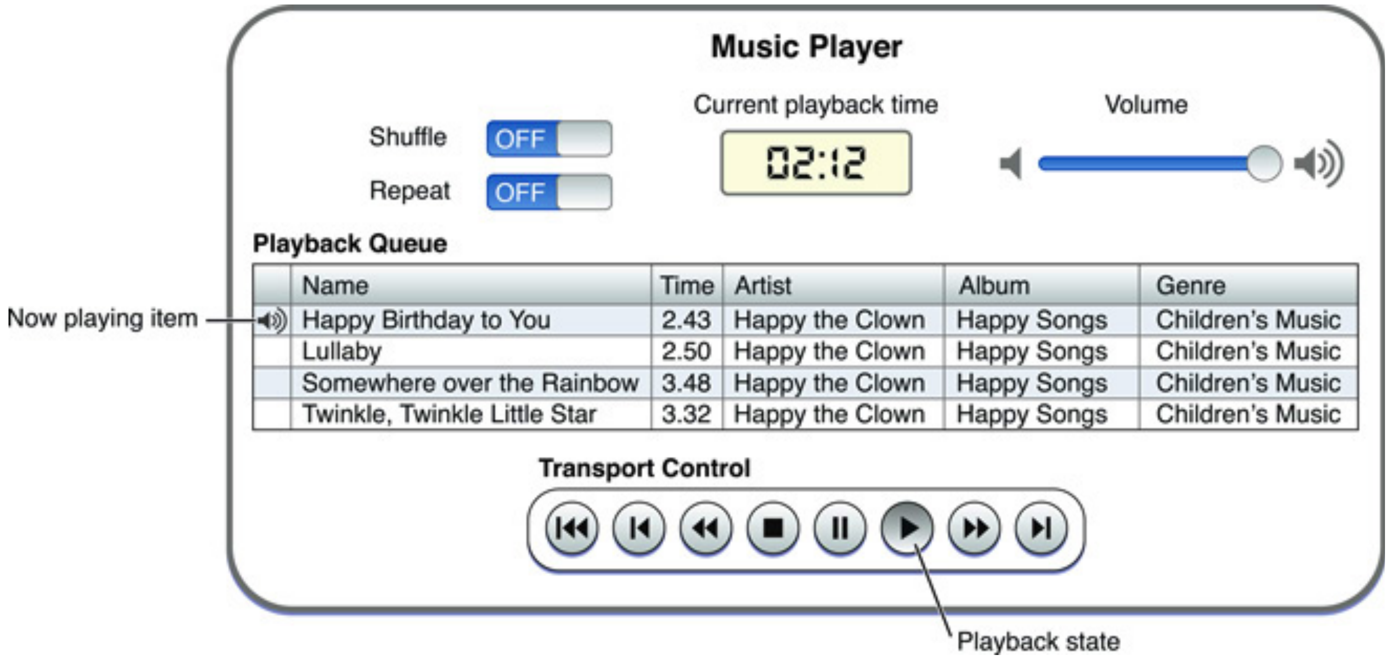
Music Player Basics and Terminology

A *music player* is the object you use for playing media items. It has a *playback queue*, which is a list of media items to play. A *media item* is a song, audio podcast, or audio book.

Media items get onto a device when a user syncs them from iTunes on the desktop. On the device, the complete set of media items is called the *iPod library*.

A music player knows which item is now playing or designated to play, and knows where in that item's timeline playback is at a given moment. These attributes are depicted schematically in Figure 1-2. (The figure is for explanation only. A music player does not provide a user interface.)

Figure 1-2 Schematic representation of a music player object



	Name	Time	Artist	Album	Genre
🔊	Happy Birthday to You	2.43	Happy the Clown	Happy Songs	Children's Music
	Lullaby	2.50	Happy the Clown	Happy Songs	Children's Music
	Somewhere over the Rainbow	3.48	Happy the Clown	Happy Songs	Children's Music
	Twinkle, Twinkle Little Star	3.32	Happy the Clown	Happy Songs	Children's Music

The *now playing* item has special status. For example, if a user pauses a song in the built-in iPod application, and then launches your application, your music player can continue playing that item from the same spot.

Additional properties round out the music player, making it highly flexible. As shown in the figure, a music player also has modes, playback state, and volume.

- *Shuffle mode* and *repeat mode* operate the same way they do in the built-in iPod app.

- *Playback state* is what you'd expect from any audio playback system: playing, stopped, paused, fast forward, or rewind. In addition, you can skip to the beginning of the next or previous media item and return to the start of a playback queue.
- *Volume* is full by default and can be set to any value down to silent.

You can obtain two flavors of music player, depending on the goals of your application.

- The *application music player* plays music locally within your application. It is not influenced by, nor does it affect, the state of the built-in iPod application. Specifically, your music player has a different now-playing item, playback state, and modes. When a user quits your application, music that you are playing stops.
- The *iPod music player*, in effect, employs the built-in iPod app on your behalf; it shares the iPod's state. When a user quits an app that is using this player, the music continues to play.

Finally, keep in mind the following two important points about using music players:

- Only one music player can play audio at a time.
- A music player can be used only on the main thread of your app.

Hello Music Player

Here is a bare bones *hello world*-style example that demonstrates library access and music playback. In a few minutes you can have a working, if minimal, music player. Lacking a user interface, this code queues up the entire device iPod library and starts playing immediately on launch.

Note: To follow these steps you'll need a provisioned device because the Simulator has no access to a device's iPod library.

1. *Create a new Xcode project.*

   In Xcode, create a new project using the Window-based Application template. In the project window, add the `MediaPlayer` framework to the Frameworks group. Save the project.

2. *Import the umbrella header file for the Media Player framework.*

   Add the following line to the `AppDelegate.h` file, after the existing `#import` line:

   ```
   #import <MediaPlayer/MediaPlayer.h>
   ```

3. *Add code to create a music player, assign it music to play, and start playback.*

   Open the project's `AppDelegate.m` implementation file. Before the end of the `applicationDidFinishLaunching:` block, add the three lines of code shown in Listing 1-1.

Listing 1-1 A very bare-bones music player

```
// instantiate a music player

MPMusicPlayerController *myPlayer =
```

```
    [MPMusicPlayerController applicationMusicPlayer];


// assign a playback queue containing all media items on the device
[myPlayer setQueueWithQuery: [MPMediaQuery songsQuery]];


// start playing from the beginning of the queue
[myPlayer play];
```

Now, configure your project appropriately for your development device, which includes setting the code-signing identity and the bundle identifier. Also, ensure that the device has at least one song in its iPod library. Build and run the project. When the application launches on the device, the first song in the iPod library starts playing. The player continues to play through all the items in the iPod library or until you quit the application.

About Music Player Change Notifications

To keep track of what a music player is doing, you register for music player change notifications. This is essential for ensuring that your application's state and the music player's state are well coordinated.

For example, here is the sequence of events for correctly starting up music playback. Notice that, because music players operate on their own threads, you do not update your user interface until receiving the appropriate notification.

1. A user taps Play.
2. Your application invokes playback on the music player.
3. The music player starts playing and issues a playback-state-change notification.
4. Your application receives the notification and queries the music player's state, confirming that it is indeed playing.
5. Your application updates its user interface accordingly—perhaps changing the Play button to say Pause.

Music player change notifications support keeping track of playback state, the now-playing item, and the music player's playback volume. Using Media Playback explains how to use them.

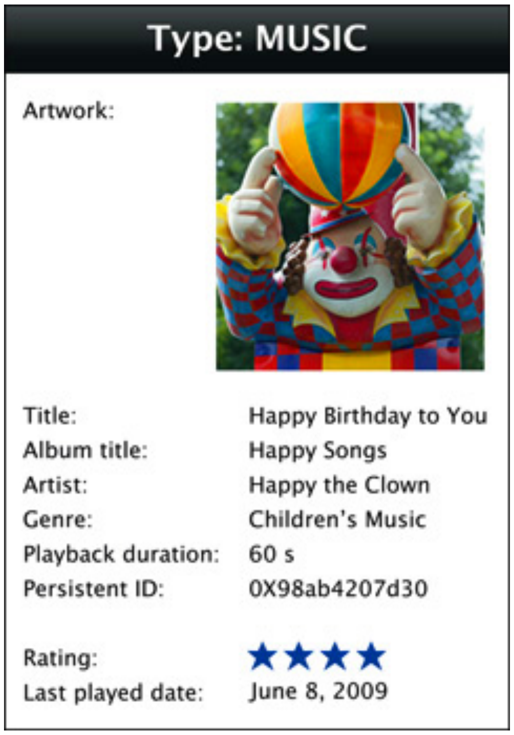
Home Sharing and iPod Music Players

Starting in iOS 4, the built-in iPod and Videos apps can play media from shared libraries using Home Sharing. However, third-party apps using the Media Player framework still have access only to the device iPod library. This means that your app cannot display the title of a home-shared song in your user interface. Other playback information—such as current playback time and playback state—is available, however, when playing shared media.

About Media Items and the iPod Library

Media items—the songs, audio books, and audio podcasts in the iPod library—can have a wide range of metadata. You can use this metadata in building queries and in creating attractive displays of the media items in the user interface. Figure 1-3 gives you an idea of the character of a media item.

Figure 1-3 A media item and some of its metadata



Type: MUSIC

Artwork:	
Title:	Happy Birthday to You
Album title:	Happy Songs
Artist:	Happy the Clown
Genre:	Children's Music
Playback duration:	60 s
Persistent ID:	0X98ab4207d30
Rating:	★★★★
Last played date:	June 8, 2009

All media item metadata is read-only. However, by using media item Persistent ID values, you can associate additional metadata that you manage in your application.

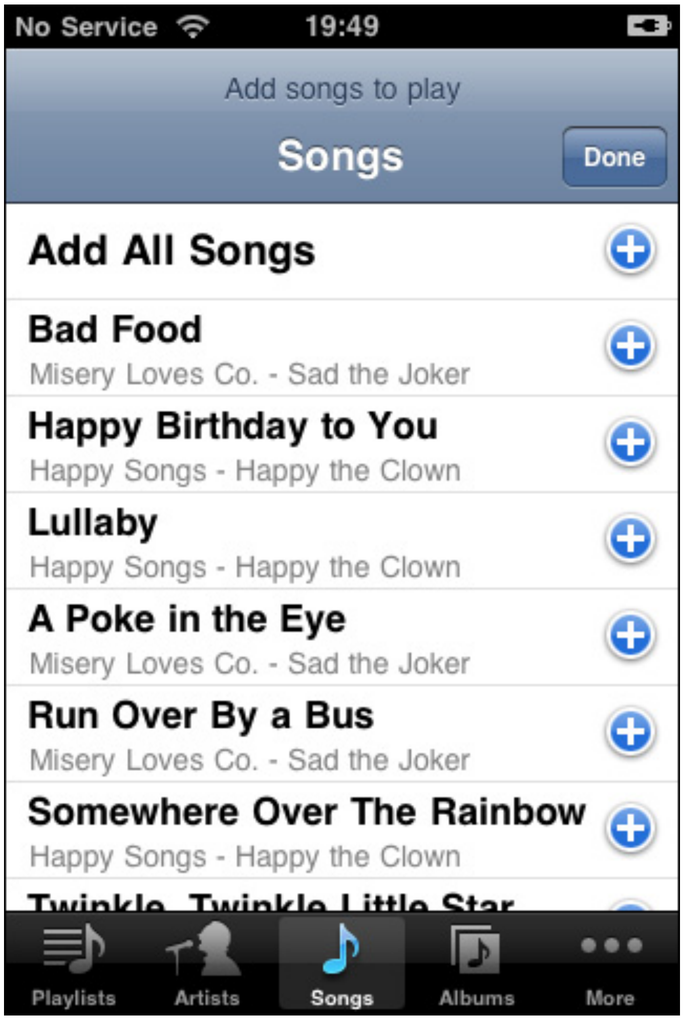
As Figure 1-3 suggests, an item's metadata can include more than one category of information. So-called *general properties* are those that can apply to any media item. These include "title," "artist," "artwork," and many others. The values of these properties typically do not change over time.

A media item can also have *user-defined properties* such as rating and last-played date. These properties update according to user activity, just as they do on the desktop.

The Media Item Picker

The simplest way to enable users to choose music is to use a *media item picker*—a fully-configured modal view controller. Its user interface is similar to the built-in iPod application's on-the-go interface, as shown in Figure 1-4.

Figure 1-4 The user interface of the media item picker


No Service
19:49
Add songs to play
Songs
Done
Add All Songs
Bad Food
Misery Loves Co. - Sad the Joker
Happy Birthday to You
Happy Songs - Happy the Clown
Lullaby
Happy Songs - Happy the Clown
A Poke in the Eye
Misery Loves Co. - Sad the Joker
Run Over By a Bus
Misery Loves Co. - Sad the Joker
Somewhere Over The Rainbow
Happy Songs - Happy the Clown
Twinkle, Twinkle Little Star
Playlists
Artists
Songs
Albums
More

When the user taps Done, a delegate method that you implement receives the list of chosen media items and then dismisses the picker's view.

It's important to know that the similarity between the media item picker and the built-in iPod app is only skin deep. Using the iPod, a user can build an on-the-go playlist. That playlist acts like other playlists; for example, it appears in the Playlists tab of the iPod application and it persists until the user changes or deletes it.

With the picker, a user instead specifies a *collection* of media items. The collection does not have the status of a playlist. It won't persist after the user quits your application—unless you archive it. Neither will the collection appear, under any circumstances, in the Playlists tab of the picker.

Getting Media Items Programmatically

If the media item picker doesn't provide the control you want, you can use the database access classes from this API. These classes are designed to let you create arbitrarily complex queries. You could, for example, retrieve all the songs in a particular genre whose titles include a particular word or phrase.

Using programmatic access is a two step process:

1. Configure a query.

2. Ask the query to retrieve its matching media items.

A *media query* is a description of what to retrieve from the device iPod library and how those retrieved items should be arranged. It has two properties to configure:

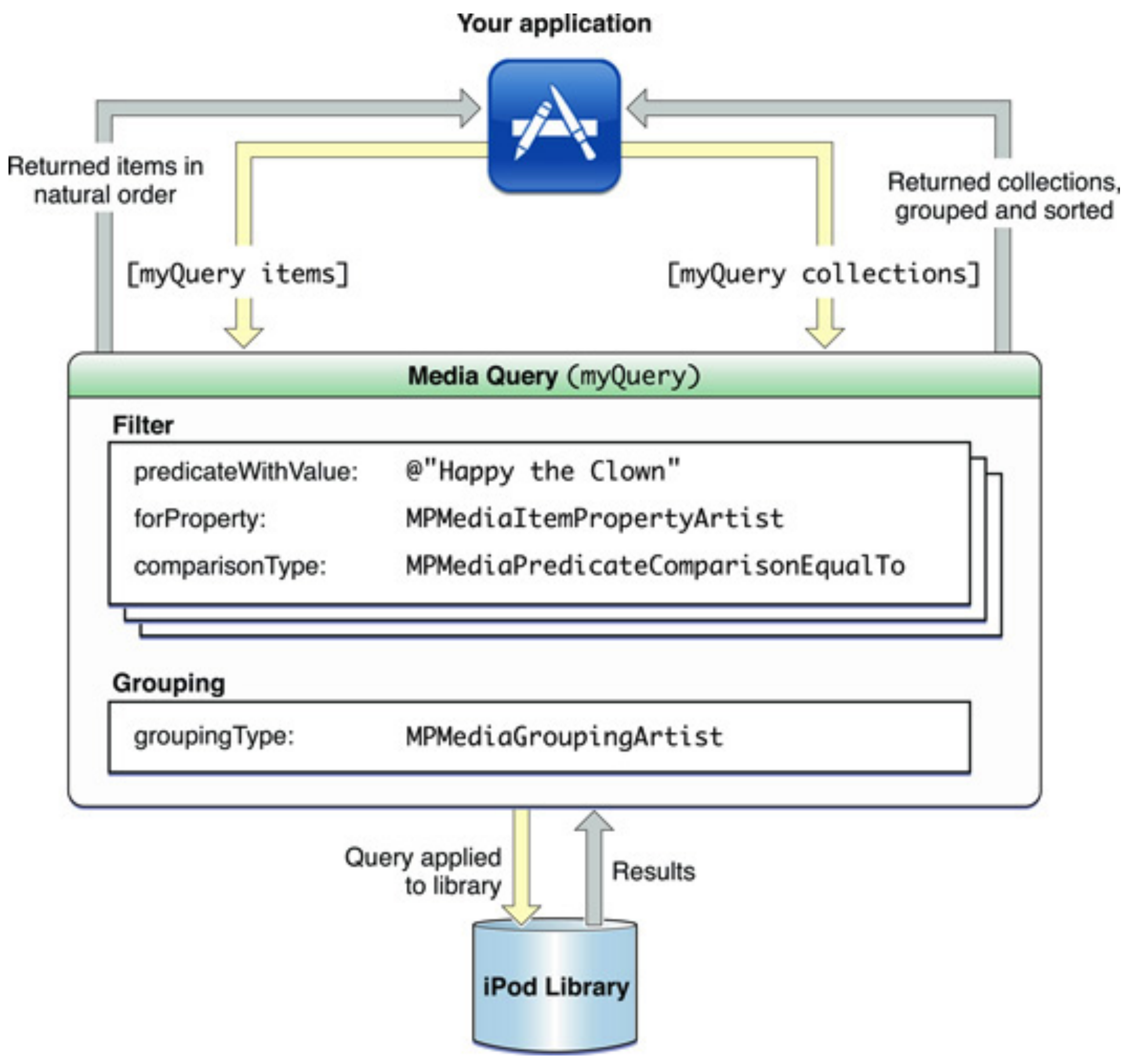
- The *filter* is the description of what to retrieve. The filter is optional; a filterless query matches the entire iPod library.
- The *grouping type* is an optional key that specifies the arrangement to use for retrieved collections of media items.

Zooming in a bit more, the filter can be as simple or complex as your application demands. It consists of one or more instances of a media property predicate. A *media property predicate* is a statement of a logical condition to test each media item against. The items that satisfy the filter are retrieved from the iPod library when you invoke the query.

The optional grouping type specifies the arrangement and sorting of collections as well as the sorting of media items within each collection. For example, using an "album" grouping type results in returned media items grouped by album, with each album's songs sorted in track order.

Figure 1-5 shows a configured media query and its place between your application and the iPod library.

Figure 1-5 Using a media query to access the device iPod library


Your application
Returned items in natural order
Returned collections, grouped and sorted
[myQuery items]
[myQuery collections]
Media Query (myQuery)
Filter
predicateWithValue: @"Happy the Clown"
forProperty: MPMediaItemPropertyArtist
comparisonType: MPMediaPredicateComparisonEqualTo
Grouping
groupingType: MPMediaGroupingArtist
Query applied to library
Results
iPod Library

As the figure shows, a query can fetch items or collections.

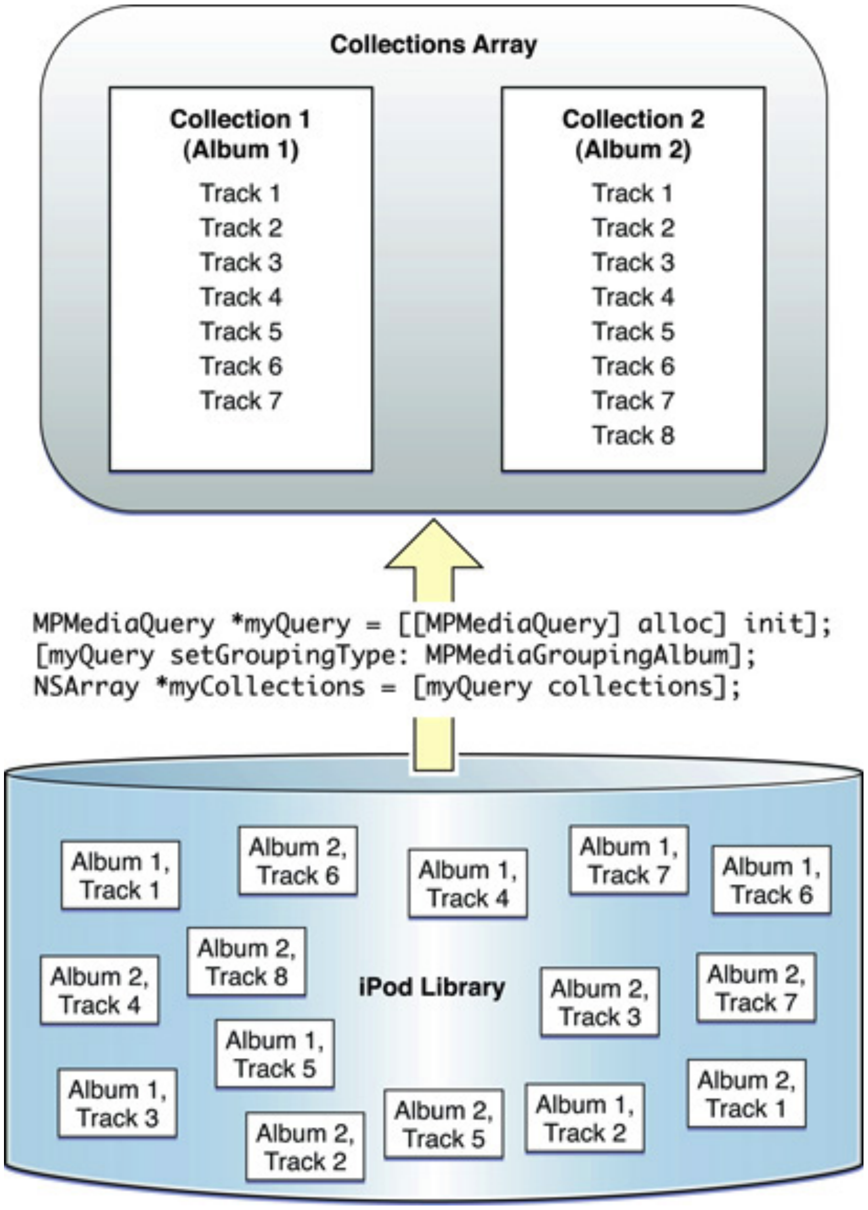
- When you ask for items, the query returns a collection containing all the items that match the filter. The items are in “natural” order, meaning that they are ordered as iTunes shows them on the desktop.
- When you ask for collections, the media query employs not only its filter but also its grouping type.

About Collections and Playlists

A *media item collection* is an array of media items. You obtain collections from the device iPod library by accessing a media query’s `collections` property. The returned value is a sorted array of collections where each collection is an instance of the query’s grouping type.

For example, say you specify an “album” grouping type by assigning the `MPMediaGroupingAlbum` key to a media query—and say that query has no filter. The value of the `collections` property is then the entire audio content of the iPod library arranged as albums, one album per collection. The collections (albums in this case) are sorted alphabetically. The songs (each a media item) within each collection are sorted by track number. Figure 1-6 illustrates this.

Figure 1-6 Obtaining collections from the device iPod library



```
MPMediaQuery *myQuery = [[MPMediaQuery] alloc] init];
[myQuery setGroupingType: MPMediaGroupingAlbum];
NSArray *myCollections = [myQuery collections];
```

At the bottom of the figure you see the iPod library for a device. In this illustrative case, the library has just two music albums, one with seven tracks and the other with eight. The items are in their “natural” order, as they appear in iTunes on the desktop.

The up-pointing arrow in the middle of the figure represents the definition of a generic (filterless) query, the application of the “album” grouping type, and the accessing of that query’s `collections` property.

The top of the figure represents the result of the `collections` call. It is an array whose elements are, in turn, arrays of media items. The collections array is sorted by album name. Each collection is sorted by track number.

You can construct your own collections as well. This can be useful, for example, for managing the selections a user has made with a media item picker. Note, however, that collections are not mutable.

A *playlist* is special sort of media item collection. It has properties that regular collections do not have, such as a name and a persistent ID. Playlists are created by users on the desktop; on the device they are read-only.

Using iPod Library Change Notifications

The `MPMediaLibrary` object represents the state of the device iPod library. You can use it to ensure that your cache of the library contents is up to date. This is useful because a user may sync their device, changing the content of the iPod library, while your application is running.

Mixing Your Own Sounds with iPod Library Sounds

Note: To mix application sounds with sounds from the device iPod library, you need to understand audio sessions, audio session categories, audio hardware route changes, and audio interruptions. To learn about these, refer to *Audio Session Programming Guide*.

You also need to understand how your choice of audio formats can impact the simultaneous playback of sounds on a device. See Playing Multiple Sounds Simultaneously in *Core Audio Overview*.

A music player automatically employs the Media Playback audio session category. If your application uses a music player and no other sounds, you should write no audio session code. Specifically, you should not initialize, configure, or activate the audio session. The system automatically handles playback, audio hardware route changes, and audio interruptions for music players.

On the other hand, to mix application sounds with iPod library sounds, you need to configure and use your application's audio session. Employ the Ambient category to support mixing. Handle audio interruptions, audio hardware route changes, and audio session reactivation as described in *Audio Session Programming Guide*. For an example of mixing application sound with iPod sound, see the *AddMusic* sample.